



AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

**WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**

KATEDRA INFORMATYKI STOSOWANEJ

Praca dyplomowa inżynierska

*Implementacja systemu uwierzytelniania z zastosowaniem
Negatywnych Baz Danych*

Implementation of authentication system using Negative Databases

Autor:

Grzegorz Nieużyła

Kierunek studiów:

Informatyka

Opiekun pracy:

dr inż. Piotr Szwed

Kraków, 2020

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.”, a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».”, oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Spis treści

1. Wprowadzenie	7
1.1. Cele pracy	7
1.2. Zawartość pracy	7
2. Negatywne Bazy Danych - opis teoretyczny	9
2.1. Opis działania	9
2.2. Zastosowanie w systemach uwierzytelniania	9
2.3. Algorytmy generacji Negatywnych Baz Danych	10
2.3.1. Algorytm prefiksowy	10
2.3.2. Algorytm <i>Randomize_NDB</i>	11
2.3.3. Algorytm 0-Hidden	15
2.3.4. Algorytmy 1-Hidden i 2-Hidden	15
2.3.5. Algorytm Q-Hidden	16
3. Solwery SAT	17
3.1. Opis działania	17
3.2. Wykorzystywanie solwerów SAT w celu uzyskania przeciwobrazu Negatywnej Bazy Danych	17
4. Implementacja systemu uwierzytelniania	19
4.1. Reprezentacja danych	19
4.2. Algorytm tworzenia użytkownika	19
4.3. Algorytm uwierzytelniania użytkownika	19
4.4. Działanie aplikacji	19
5. Testy implementacji	21
5.1. Opis testowania za pomocą solwerów SAT	21
5.2. Testy algorytmów prostych	21
5.3. Testy algorytmów złożonych	21
6. Wnioski	23
6.1. Bezpieczeństwo przedstawionej implementacji	23

6.2. Możliwe rozszerzenia.....	23
----------------------------------	----

1. Wprowadzenie

1.1. Cele pracy

Celem niniejszej pracy jest implementacja i przetestowanie systemu uwierzytelniania oferującego większe bezpieczeństwo niż standardowy schemat generowania skrótu hasła za pomocą funkcji generacji klucza (np. PBKDF2, bcrypt) i przechowywaniu w standardowej (pozytywnej) bazie danych.

Założeniem systemu jest zamienienie reprezentacji w sposób jawny na Negatywną Bazę Danych (NDB) co pozwoli dodać dodatkową warstwę bezpieczeństwa która znacząco utrudni uzyskanie haseł użytkownika w przypadku wykradzenia bazy danych.

Rezultatem wyjściowym algorytmów generacji NDB jest zbiór formuł logicznych, dlatego przeprowadzone zostały testy z wykorzystaniem solverów SAT mające na celu zasymulowanie ataku na powstałą NDB.

1.2. Zawartość pracy

2. Negatywne Bazy Danych - opis teoretyczny

2.1. Opis działania

Główną operacją wykonywalną na NDB jest sprawdzenie czy dany rekord znajduje się w bazie. Przyjmując U jako oznaczenie uniwersum języka binarnego o długości l a DB jako zbiór wszystkich rekordów, każdy o długości l , NDB przechowuje zbiór $U - DB$ [1]. Takie dane są niepraktycznie do zareprezentowania w postaci nieskompresowanej z uwagi na wielkość, dlatego stosuje się wyrazy nad alfabetem $\{0, 1, *\}$ gdzie symbol $*$ może oznaczać zarówno 0 lub 1 w jawnej reprezentacji bitowej. Pozycje na których znajduje się wartość 0 lub 1 są *ustalone* a z wartością $*$ - *nieustalone*.

Każdy taki wyraz odpowiada jednemu lub wielu elementom $U - DB$ i jest sprowadzany do formuły logicznej (Tabela 2.1). Z założenia algorytm sprawdzający przynależność do DB sprawdza czy jakakolwiek formuła z NDB jest spełniana przez dany rekord. Dane znajdują się w DB wtedy i tylko wtedy gdy żadna formuła nie zostanie spełniona.

Taki model działania wymusza na danych stałą wielkość, co jednak nie stanowi problemu w przypadku przechowywania skrótów haseł które mają stałą, zależną od konkretnego algorytmu długość. Dla danych o zmiennych rozmiarach (np. nazwy użytkownika) można zastosować funkcję hashującą lub algorytmy zwiększające długość ciągu bitowego do stałej wartości typu PKCS#5 lub PKCS#7. Należy jednak pamiętać, że zwiększenie długości rekordu znacznie wydłuża czas generacji bazy oraz zajmowaną pamięć.

Tabela 2.1. Reprezentacja formuł logicznych za pomocą NDB

rekord NDB	formuła logiczna
011*	$\neg x_1 \wedge x_2 \wedge x_3$
0*01	$\neg x_1 \wedge \neg x_3 \wedge x_4$
111*	$x_1 \wedge x_2 \wedge x_3$

2.2. Zastosowanie w systemach uwierzytelniania

NDB może być wykorzystana w każdym systemie, gdzie podstawową operacją na danych jest sprawdzenie czy dany rekord znajduje się w bazie. Jednym z najpopularniejszych systemów uwierzytelniania

jest metoda oparta na loginie i hasle. Użytkownik danej aplikacji przy zakładaniu konta podaje hasło, które następnie warstwa serwerowa danej aplikacji przechowuje jako wynik nieodwracalnej funkcji hashującej.

W przypadku nieautoryzowanego dostępu do bazy danych i używanego algorytmu uzyskiwania skrótu z hasła, atakujący może uzyskać wartość pierwotną mało skomplikowanych haseł za pomocą np. metody słownikowej. Modyfikując powyższy algorytm składując skróty jako rekordy w NDB uniemożliwiamy iterację wszystkich danych, jednocześnie pozostawiając łatwy dostęp do informacji czy użytkownik o podanym loginie i hasle ma dostęp do aplikacji.

2.3. Algorytmy generacji Negatywnych Baz Danych

2.3.1. Algorytm prefiksowy

Najprostszym ze sposobów generowania Negatywnych Baz Danych jest zaproponowany przez Fernando Esponda algorytm prefiksowy [1, 2]. Został on opracowany w celu udowodnienia że proces generowania NDB z rekordów DB jest możliwy w rozsądnej złożoności czasowej i pamięciowej.

Algorithm 1: Algorytm prefiksowy

Data: DB - zbiór rekordów do zareprezentowania w NDB, l - liczba rekordów w DB

Result: Zbiór rekordów NDB

```

1  $Prefix_n(V)$  - Prefiks  $n$ -znakowy rekordu  $V$ 
2  $len(V)$  - Długość rekordu  $V$ 
3  $W_i = \{\}$ ;
4  $i = 0$ ;
5 while  $i < l$  do
6    $W_{i+1} =$  Zbiór wszystkich  $i + 1$ -znakowych ciągów bitowych  $V_p$  nie będących prefiksem
      żadnego rekordu  $DB$  i dla których  $Prefix_i(V_p) \in W_i$ 
7   for  $V_p$  in  $W_{i+1}$  do
8     Stwórz rekord NDB o długości  $l$  którego  $V_p$  jest prefiksem a na pozostałych pozycjach
      jest symbol * i dodaj do zbioru wyjściowego NDB
9   end
10   $i = i + 1$ ;
11   $W_i =$  Zbiór wszystkich  $i$ -znakowych prefiksów rekordów  $DB$ 
12 end
```

Powyższa metoda polega na generowaniu coraz dłuższych prefiksów które nie pokrywają się ze zbiorem DB . W ten sposób na początku tworzone są rekordy odpowiadające znacznej części $U - DB$. Czasami występuje potrzeba zdefiniowania pewnych rekordów explicite bez wykorzystania symbolu $*$ jeżeli każdy możliwy prefiks jest także prefiksem rekordu z DB . Przykładowy wynik działania znajduje się w tabeli 2.2.

DB	U - DB	NDB
0000	0001	10**
0110	0011	010*
0010	0100	111*
1101	0101	0001
	0111	0011
	1000	0111
	1001	1100
	1010	
	1011	
	1100	
	1110	
	1111	

Tabela 2.2. Rezultat działania algorytmu prefiksowego

Algorytm prefiksowy jest deterministyczny i każdy powstały rekord reprezentuje unikalną, nie pokrywającą się część $U - DB$ [1]. Powoduje to, że algorytm uzyskiwania zbioru DB z otrzymanej NDB nie wymaga sprowadzenia do problemu SAT. Wystarczy jedynie odpowiednio posortować rekordy i wyznaczyć przedziały pomiędzy nimi.

Czas wykonywania procedury wynosi $O(l|DB|)$, jednak w przypadku zapisywania wyniku do bazy wzrasta do $O(l^2|DB|)$ gdyż konieczna jest serializacja każdego rekordu. Złożoność obliczeniowa dla optymalnej implementacji wynosi $O(l|DB|)$ w przypadku gdy poprzednio generowane rekordy NDB nie są przetrzymywane w pamięci. Dla danego zbioru DB generowane jest $O(l|DB|)$ rekordów co sprowadza się do wielkości powstałej bazy danych wynoszącej $O(l^2|DB|)$.

2.3.2. Algorytm *Randomize_NDB*

Algorytm prefiksowy generuje poprawne rekordy NDB, jednak nie jest praktyczny w żadnych zastosowaniach związanych z bezpieczeństwem, ponieważ wynik jego działania jest stosunkowo prosto

sprowadzić do postaci pozytywnej. Aby temu zaradzić, Fernando Esponda w swojej pracy[1] zaproponował niedeterministyczny algorytm mający na celu wprowadzić rekordy które nie odpowiadają jedynie prefiksom elementów z $U - DB$ i są trudniejsze do odwrócenia.

Algorithm 2: Algorytm Randomize_NDB

Data: DB - zbiór rekordów do zarepresentowania w NDB, l - liczba rekordów w DB

Result: Zbiór rekordów NDB

```

1  $Prefix_n(V)$  - Prefiks  $n$ -znakowy rekordu  $V$ 
2  $len(V)$  - Długość rekordu  $V$ 
3  $\pi$  = losowa permutacja o długości  $|V_{pe}|$ 
4  $W_i$  = zbiór wszystkich ciągów  $l$ -bitowych
5  $\pi(DB) \equiv \{\pi(V) \mid V \in DB\}$ 
6  $i = \lceil \log_2(l) \rceil$ ;
7 while  $i < l$  and  $W_i \neq \emptyset$  do
8    $W_{i+1}$  = Zbiór wszystkich  $i + 1$ -znakowych ciągów bitowych  $V_p$  nie będących prefiksem
      żadnego rekordu  $\pi(DB)$  i dla których  $Prefix_i(V_p) \in W_i$ 
9   for  $V_p$  in  $W_{i+1}$  do
10     Dopełnij  $V_p$  do długości  $l$  wstawiając znaki nieustalone '*' na końcu
11      $j$  = losowa liczba z przedziału  $[1, l]$ 
12     for  $k = 1$  to  $j$  do
13        $n$  = losowa liczba z przedziału  $[1, \log_2(l)]$ 
14        $P = n$  losowych nieustalonych pozycji z  $V_p$ 
15        $X$  = Zbiór rekordów powstałych przez zastąpienie pozycji  $\in P$  przez wszystkie
        możliwe kombinacje bitowe ( $2^n$  rekordów)
16       for  $V_q$  in  $X$  do
17          $V_{pg} = \text{Pattern\_Generate}(\pi(DB), V_q)$ 
18         Dodaj  $\pi^{-1}(V_{pg})$  do zbioru rekordów wyjściowych
19       end
20     end
21   end
22    $i = i + 1$ 
23    $W_i$  = Zbiór wszystkich  $i$ -znakowych prefiksów rekordów  $DB$ 
24 end

```

Algorytm ten działa na podobnej zasadzie co algorytm prefiksowy (rozdział 2.3.1) z pewnymi modyfikacjami. Na początku kolejność bitów w DB jest mieszana za pomocą losowej permutacji aby pozycje zdefiniowane w generowanej prefiksowej NDB nie były skumulowane na początku wyrazów. Następnie dla każdego powstałego negatywnego rekordu losuje się n pozycji nieustalonych i zastępuje

się go równoznacznym zbiorem rekordów które mają te pozycje ustalone. Powstałe ciągi są dodatkowo obfuskowane przez wstawienie na losowych pozycjach zamiast bitu zdefiniowanego znak * zgodnie z algorytmem *Pattern_Generate*.

Wynik algorytmu jest niedeterministyczny co powoduje że dla takich samych zbiorów *DB* rezultat może się różnić. Generacja wielu redundantnych rekordów zwiększa odporność otrzymanej bazy na próby przywrócenia do postaci pozytywnej, jednak wiąże się to z ze zwiększeniem objętości NDB średnio $\frac{l^2}{2}$ razy w stosunku do algorytmu prefiksowego.

Algorithm 3: Algorytm *Pattern_Generate*

Data: *DB* - zbiór rekordów do zarepresentowania w NDB, V_{pe} - rekord NDB do zobfuskowania

Result: Zobfuskowany rekord NDB

```

1  $\pi$  = losowa permutacja o długości  $|V_{pe}|$ 
2  $SIV = \{\}$  // wektor zmienionych bitów
3 for  $i = 1$  to  $|V_{pe}|$  do
4    $\pi(V_{pe})' = \pi(V_{pe})$  z elementem o indeksie  $i$  zamienionym na symbol *
5   if istnieje rekord w DB pokrywający się z  $\pi(V_{pe})'$  then
6     Dodaj  $i$  i bit o indeksie  $i$  do  $SIV$ 
7      $\pi(V_{pe}) = \pi(V_{pe})'$ 
8   end
9 end
10  $t$  = losowa liczba z przedziału  $[0, |SIV|]$ 
11 if  $t > |SIV|$  then
12    $R = SIV$ 
13 else
14    $R = t$  losowych bitów z  $SIV$ 
15 end
16  $V_k = \pi(V_{pe})$ 
17 for indeks, bit in  $R$  do
18    $V_k[indeks] = bit$ 
19 end
20 Zwróć  $\pi^{-1}(V_k)$ 

```

Zmiany względem algorytmu prefiksowego oraz procedura *Pattern_Generate* gwarantują że żaden ze zmodyfikowanych rekordów nie będzie odpowiadał żadnemu elementowi *DB* oraz że wynik działania pozostanie kompletnym odzwierciedleniem $U - DB$, ponieważ każda modyfikacja rekordu rozszerza zakres pokrywanych ciągów wprowadzając redundantne informacje. W [1] przedstawiony jest dowód że problem rekonstrukcji *DB* z *NDB* jest NP-trudny (każdą instancję 3-SAT można sprowadzić do *NDB*).

Powyższy algorytm w teorii jest zdolny do tworzenia trudnych instancji, ale nie posiada żadnych mechanizmów umożliwiających ingerencję w jego działanie - wynik jest zawsze losowy. W praktyce powoduje to, że można oczekiwać, że wynikowa *NDB* może być odwrócona niemal natychmiast przez współczesne solwery SAT, co pokazują w dalszych rozdziałach.

DB	U - DB	NDB
0000	0001	*001
0110	0011	00*1
0010	0100	0011
1101	0101	010*
	0111	0*11
	1000	1**0
	1001	10**
	1010	101*
	1011	**11
	1100	1*1*
	1110	1100
	1111	111*

Tabela 2.3. Rezultat działania algorytmu *Randomize_NDB*

2.3.3. Algorytm 0-Hidden

Następujące algorytmy powstały przez zastosowanie procedur generowania trudnych instancji SAT. Jedną z nich jest **0-Hidden** służąca do generacji formuł 3-SAT nie posiadających rozwiązania, co pozwala na testowanie solverów pod względem wykrywania braku spełnialności [3].

Algorithm 4: Algorytm 0-Hidden

Data: l - liczba zmiennych, r - współczynnik ilości klauzul

Result: Zbiór klauzul 3-SAT

```

1  $n = l * r$ 
2  $W = \{ \}$ 
3 while  $|W| \neq n$  do
4   Wybierz 3 losowe zmienne
5   Stwórz klauzulę 3CNF używając wylosowane zmienne, z losowymi znakami
6   Dodaj klauzulę do zbioru wynikowego  $W$ 
7 end
```

Powyższy algorytm generuje formułę CNF, która z dużym prawdopodobieństwem nie jest spełnialna i jest trudna do rozwiązania przez solwery SAT [3, 4]. Modyfikując parametr r możemy wpłynąć na rozmiar formuły, wartość $r \approx 4.27$ jest wartością graniczną powyżej której problem prawie na pewno nie ma rozwiązania [3].

Z perspektywy Negatywnych Baz Danych można przyjąć że powstała formuła odpowiada zbiorowi $U - DB$, jeśli DB jest zbiorem pustym.

2.3.4. Algorytmy 1-Hidden i 2-Hidden

Algorytmy **1-Hidden** i **2-Hidden** są skonstruowane podobnie jak 0-Hidden, z tą różnicą, że formuła wyjściowa ma odpowiednio co najmniej (i w znacznej większości dokładnie) jedno lub dwa rozwiązania.

W celu testowania możliwości solverów SAT co do znajdowania przypisania spełniającego daną formułę można generować losowe formuły (alg. 4) i odrzucać klauzule które przeczą ukrytemu rozwiązaniu A (1-Hidden). Jednak spowoduje to, że rozkład losowy zostanie zaburzony i solwery mogą to "poczuć", co doprowadzi je do ukrytego rozwiązania [4].

Aby temu przeciwdziałać można jednocześnie ukryć przypisania A oraz $\neg A$ (2-Hidden), co spowoduje że algorytm przeszukujący będzie równoważnie "przyciągany" przez dwa przeciwne rozwiązania.

Powyższy schemat działania jest wykorzystany w procedurze **Q-Hidden** będącej rozszerzeniem tego konceptu.

2.3.5. Algorytm Q-Hidden

3. Solwery SAT

3.1. Opis działania

3.2. Wykorzystywanie solwerów SAT w celu uzyskania przeciwobrazu Negatywnej Bazy Danych

4. Implementacja systemu uwierzytelniania

4.1. Reprezentacja danych

4.2. Algorytm tworzenia użytkownika

4.3. Algorytm uwierzytelniania użytkownika

4.4. Działanie aplikacji

5. Testy implementacji

5.1. Opis testowania za pomocą solwerów SAT

5.2. Testy algorytmów prostych

5.3. Testy algorytmów złożonych

6. Wnioski

6.1. Bezpieczeństwo przedstawionej implementacji

6.2. Możliwe rozszerzenia

Bibliografia

- [1] Fernando Esponda. "Negative Representations of Information". PhD thesis. 2005.
- [2] F. Esponda, S. Forrest i P. Helman. „Enhancing Privacy through Negative Representations of Data”. W: 2004.
- [3] Haixia Jia, Cristopher Moore i Doug Strain. „Generating Hard Satisfiable Formulas by Hiding Solutions Deceptively”. W: *Journal of Artificial Intelligence Research - JAIR* 28 (mar. 2005). DOI: *10.1613/jair.2039*.
- [4] Dimitris Achlioptas, Haixia Jia i Cristopher Moore. „Hiding Satisfying Assignments: Two are Better than One”. W: *Proceedings of the National Conference on Artificial Intelligence* 24 (kw. 2005). DOI: *10.1613/jair.1681*.